Università della Svizzera italiana

Institute of Computing CI

High-Performance Computing Lab **Institute of Computing**

Student: Marenco Turi Gualtiero Discussed with: Sobolev Mark,

Solution for Project 7

HPC Lab — Submission Instructions
(Please, notice that following instructions are mandatory: submissions that don't comply with, won't be considered)

- Assignments must be submitted to iCorsi (i.e. in electronic format).
- Provide both executable package and sources (e.g. C/C++ files, Matlab). If you are using libraries, please add them in the file. Sources must be organized in directories called:
  Project_number_lastname_firstname
  and the file must be called:
  project_number_lastname_firstname.zip
  project_number_lastname_firstname.pdf
- The TAs will grade your project by reviewing your project write-up, and looking at the implementation you attempted, and benchmarking your code's performance.
- You are allowed to discuss all questions with anyone you like; however: (i) your submission must list anyone you discussed problems with and (ii) you must write up your submission independently.

1. The project tasks [in total 85 points]

1.1. Discretization

The problem is descritized by building a uniform grid with n_x and n_y points respectively in the x_1 - and x_2 -directions. For each grid point (i, j), the Laplace operator is approximated using centered finite differences of the second-orders:

$$\Delta u \approx \frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2},$$

where:

$$\frac{\partial^2 u}{\partial x_1^2} \approx \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta x^2}, \quad \frac{\partial^2 u}{\partial x_2^2} \approx \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta y^2}.$$

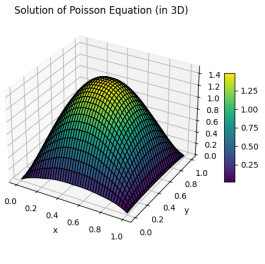
The discrete difference approximation of $-\Delta u = f$ becomes:

$$\left(\frac{2}{\Delta x^2} + \frac{2}{\Delta y^2}\right) u_{i,j} - \frac{1}{\Delta x^2}\left(u_{i-1,j} + u_{i+1,j}\right) - \frac{1}{\Delta y^2}\left(u_{i,j-1} + u_{i,j+1}\right) = f_{i,j}.$$

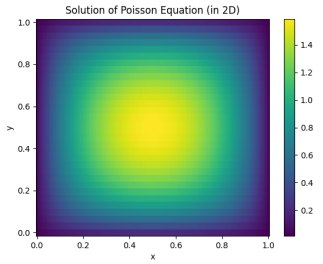
1.2. Implement the solution in Python [25 points]

1.3. Implement the solution in PETSc [25 points]

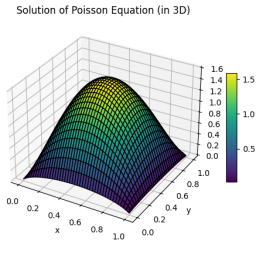
1.4. Validate and visualise the solution [10 points]



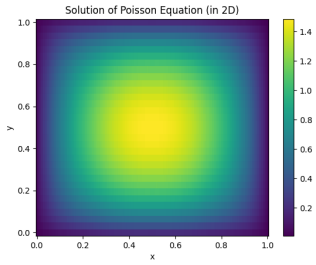
(a) Sp solver 3d



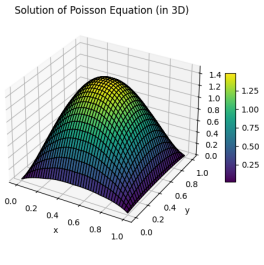
(b) Petsc 2d



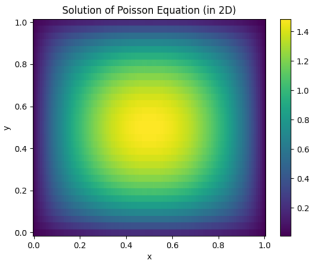
(c) Petsc 3d



(d) Dn solver 2d in python



(e) Dn solver 2d in python



(f) Sp solver 2d

Figure 1: Visualization results for *test_val/test.sh*

1.5. Performance Benchmark [15 points]

- T_{assembly} be the matrix assembly time.
- T_{solve} be the time to solve the linear system.

The total runtime, T_{total}, is given by:

$$T_{\text{total}} = T_{\text{assembly}} + T_{\text{solve}}$$

The following table summarizes the dataset used for benchmarking the solvers:

Mesh Size	**PETSc (s)**	**Python_sp_dir (s)**	**Python_dn_dir (s)**	**Python_sp_cg (s)**
8	0.018	0.002	0.001	0.002
16	0.009	0.005	0.012	0.004
32	0.018	0.011	0.051	0.009
64	0.083	0.056	0.251	0.044
128	0.319	0.223	–	0.175
256	1.018	0.789	–	0.598
512	3.204	2.904	–	2.086

Table 1: Runtimes (in seconds) for used solvers at different given mesh size. Missing data indicates runtime exceeded 10 seconds.

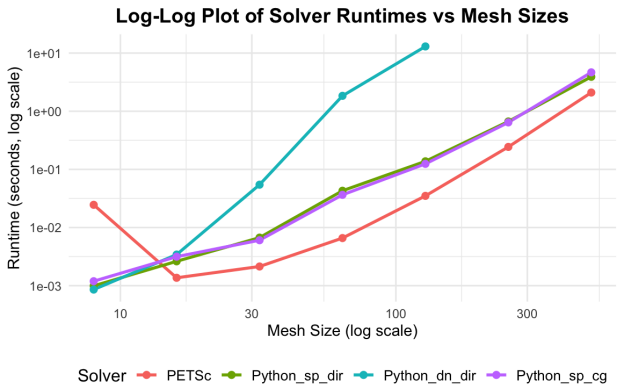


Figure 2: Serial implementation results

The first data point for PETSc has a higher runtime due to higher matrix assembly time, which become smaller for subsequent test. This is to be attributed to part of the initialization becoming stored in the cache. PETSc demonstrates excellent scaling behavior with increasing mesh sizes, outperforming the other solvers at larger grid sizes.
The sparse solver (Python_sp_dir) consistently outperforms the dense solver (`Python_dn_dir`) for medium and large grid sizes. On the opposite, the dense solver quickly becomes computationally unfit for larger workloads as reflected by the missing data points in the graph that where cut due to excessively long run time. The conjugate gradient solver (Python_sp_cg) has competitive performance, closely following PETSc's scaling behavior. While slightly slower than PETSc, its significantly outperforms the dense solver and remains suitable for larger grid sizes.
All solvers exhibit the expected trend of increasing runtimes with finer mesh grids. PETSc's superior scaling efficiency underlines its suitability for high-performance computing tasks involving large-scale problems.

1.6. Strong Scaling [10 points]

Processes	Total Runtime (seconds)	Speedup
1	19.3086	1.00
2	7.3318	2.63
4	2.5105	7.69
8	1.1382	16.96
16	0.3938	49.04

Table 2: Scaling Test Results: Total Runtime and speedup vs processes

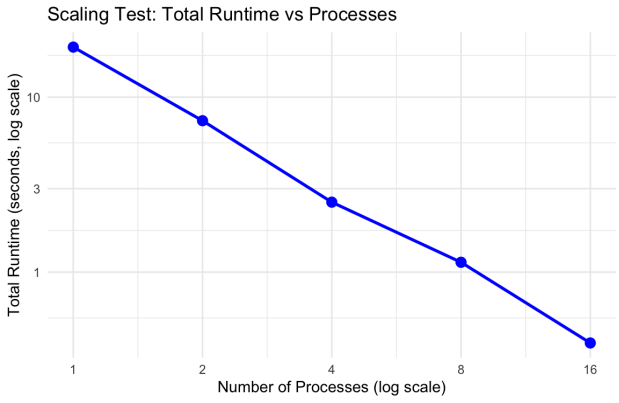


Figure 3: Speedup vs Processes scalability plot

The total runtime decreases as the number of processes increases. Remembering that it's log-log scale highlights the strong scaling behavior, with the runtime showing near-linear reduction with process count, especially for smaller process numbers. This indicates effective parallelization for this problem size.

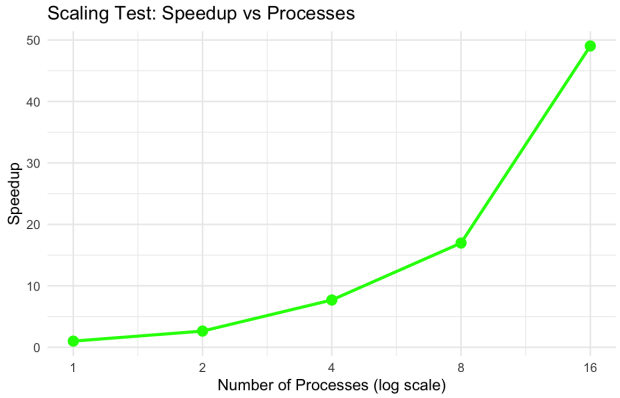


Figure 4: Computational Speedup plot

The speedup grow almost linearly with the number of processes, achieving an excellent efficiency for up to 16 processes.

The results validate the scalability of PETSc for the given grid size (which was 1024x1024), with the speedup closely following the perfect behavior.